

C# 프로그래밍

- 2주차 (2강)

문자열 다루기

- 문자열 서식화 및 출력
- 문자열 안에서 찾기
- 문자열 변형하기
- 문자열 분할하기

문자열 서식화 및 출력

- Format () 메서드

- 문자열 보간

문자열 서식 맞추기 : Format()

- string 데이터형의 메소드 (즉, `string.Format(·)`)
 - ✓ 문자열의 틀을 이용해 서식화된 새로운 문자열 생성
 - ✓ 사용 방법은 `Console.WriteLine(·)` 과 동일

```
string result = string.Format("오늘: {0}, 내일: {1}", "목", "금");  
Console.WriteLine(result);  
Console.WriteLine("오늘: {0}, 내일: {1}", "목", "금");
```

출력 결과:

오늘: 목, 내일: 금
오늘: 목, 내일: 금

- ✓ 첫 번째 매개변수에 "문자열 틀" 입력
- ✓ 두 번째 매개변수부터 문자열 틀 안에 넣을 데이터를 차례로 입력
- ✓ 문자열 틀에 입력하는 {0}, {1} .. 를 "서식항목"이라 칭함



서식항목의 다양한 서식화

- 서식 항목의 추가 옵션 구성

{첨자, 맞춤: 서식 문자열}

서식 항목의 첨자

왼쪽/오른쪽 맞춤

변환 서식 지정 문자열

```
Console.WriteLine("Total : {0, -7: D}", 123); // 첨자: 0, 맞춤: -7, 서식 문자열: D
```

- ✓ "서식 항목의 첨자"는 해당 서식항목 위치에 넣을 매개변수 지정
- ✓ "왼쪽/오른쪽 맞춤"은 서식 항목이 차지할 공간의 크기와 공간 안에서 왼쪽 또는 오른쪽에 위치 시킬지 결정
- ✓ "변환 서식 지정 문자열"은 데이터를 지정한 형태로 서식화 (예: 16진수)



왼쪽/오른쪽 맞춤

- 왼쪽 맞춤

- ✓ 서식항목이 차지할 공간의 크기를 나타내는 숫자를 음수로 표현
- ✓ 서식항목을 위한 공간 9칸 생성 후 왼쪽 맞춤 (부호 -)

```
string result = string.Format("{0,-9}DEF", "ABC");  
Console.WriteLine(result);  
// 출력 값: "ABC      DEF"
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
A	B	C							D	E	F

- 오른쪽 맞춤

```
string result = string.Format("{0, 9}DEF", "ABC");  
Console.WriteLine(result);  
// 출력 값: "      ABCDEF"
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
						A	B	C	D	E	F



왼쪽/오른쪽 맞춤 예제 코드

```
static void Main(string[] args)
{
    string fmt = "{0,-20}{1,-15}{2,30}";

    Console.WriteLine(fmt, "Publisher", "Author", "Title");
    Console.WriteLine(fmt, "Marvel", "Stan Lee", "Iron Man");
    Console.WriteLine(fmt, "Bloomsbury", "J. K. Rowling", "Harry Potter");
    Console.WriteLine(fmt, "T. Egerton", "Jane Austen", "Pride and Prejudice");
}
```

출력 결과:

Publisher	Author	Title
Marvel	Stan Lee	Iron Man
Bloomsbury	J. K. Rowling	Harry Potter
T. Egerton	Jane Austen	Pride and Prejudice



변환 서식 지정 문자열 : 수 서식화

- 다양한 형태로 수를 서식화
 - ✓ 서식 지정자와 자릿수 지정자(Precision specifier) 동시 사용 가능
 - ✓ 예: 서식 문자열 "D5" (서식 지정자: D, 자릿수 지정자: 5)

```
Console.WriteLine("{0:D5}", 123); // 출력: 00123
```

서식 지정자	대상 서식	설명
D	10진수	Console.WriteLine("{0:D}", 0xFF); // 출력: 255
X	16진수	Console.WriteLine("{0:X}", 255); // 출력: FF
N	coma(,) 표현	Console.WriteLine("{0:N}", 123456789); // 출력: 123,456,789.00
F	고정 소수점	Console.WriteLine("{0:F}", 123.45); // 출력: 123.45
E	지수	Console.WriteLine("공학: {0:E}", 123.456789); // 출력: 1.234568E+002



수 서식화 예제 코드

```
static void Main(string[] args)
{
    // D : 10진수
    Console.WriteLine("10진수: {0:D}", 123); // 10진수: 123
    Console.WriteLine(string.Format("10진수: {0:D}", 123)); // 10진수: 123
    Console.WriteLine("10진수: {0:D5}", 123); // 10진수: 00123
    // X : 16진수
    Console.WriteLine("16진수: 0x{0:X}", 0xFF1234); // 16진수: 0xFF1234
    Console.WriteLine("16진수: 0x{0:X8}", 0xFF1234); // 16진수: 0x00FF1234
    // N : 콤마 구분
    Console.WriteLine("콤마: {0:N}", 123456789); // 콤마: 123,456,789.00
    Console.WriteLine("콤마: {0:N0}", 123456789); // 콤마: 123,456,789
    // F : 고정소수점
    Console.WriteLine("고정: {0:F}", 123.45); // 고정: 123.45
    Console.WriteLine("고정: {0:F5}", 123.456); // 고정: 123.45600
    // E : 지수 표기
    Console.WriteLine("지수: {0:E}", 123.456789); // 지수: 1.234568E+002
}
```



변환 서식 지정 문자열 : 날짜 및 시간 서식화

```
DateTime dt = new DateTime(2021, 1, 8, 23, 31, 17);
Console.WriteLine(dt); // 2021-01-08 오후 11:31:17
Console.WriteLine("{0:yy-MM-dd tt hh:mm:ss (ddd)}", dt); // 21-01-08 오후 11:31:17 (금)
Console.WriteLine("{0:yyyy-MM-dd HH:mm:ss (dddd)}", dt); // 2021-01-08 23:31:17 (금요일)
```

서식 지정자	대상 서식	설명
y	연도	• yy: 두 자릿수 연도 (2021년->21), yyyy: 네 자릿수 연도 (2021년->2021)
M	월	• M: 한 자릿수 월 (1월->1), MM: 두 자릿수 월 (1월->01)
d	일	• d: 한 자릿수 일 (8일->8), dd: 두 자릿수 일 (8일->08)
h	시(1~12)	• h: 한 자릿수 시 (9시->9), hh: 두 자릿수 시 (9시->09)
H	시(1~23)	• H: 한 자릿수 일 (21시->21), HH: 두 자릿수 시 (21시->21)
m	분	• m: 한 자릿수 분 (3분->3), mm: 두 자릿수 분 (3분->03)
s	초	• s: 한 자릿수 초 (7초->7), ss: 두 자릿수 초 (7초->07)
tt	오전/오후	-
ddd	요일	• ddd: 약식 요일 (예: 토), dddd: 전체 요일 (예: 토요일)



문화권 별 날짜 및 시간 표기

- CultureInfo 클래스를 이용해 문화권 정보 표기
 - ✓ DateTime.ToString()에 서식 문자열과 이 클래스 객체 입력
 - ✓ CultureInfo 생성자에 문화권 이름은 MSDN 참조

```
static void Main(string[] args)
{
    DateTime dt = new DateTime(2021, 1, 8, 23, 31, 17);
    Console.WriteLine("{0:yyyy-MM-dd tt hh:mm:ss (ddd)}", dt);
    Console.WriteLine(dt.ToString("yyyy-MM-dd tt hh:mm:ss (ddd)"));

    CultureInfo ciKo = new CultureInfo("ko-KR");
    Console.WriteLine(dt.ToString("yyyy-MM-dd tt hh:mm:ss (ddd)", ciKo));
    Console.WriteLine(dt.ToString(ciKo));

    CultureInfo ciEn = new CultureInfo("en-US");
    Console.WriteLine(dt.ToString("yyyy-MM-dd tt hh:mm:ss (ddd)", ciEn));
    Console.WriteLine(dt.ToString(ciEn));
}
```

출력 결과:

```
2021-01-08 오후 11:31:17 (금)
2021-01-08 오후 11:31:17 (금)
2021-01-08 오후 11:31:17 (금)
2021-01-08 오후 11:31:17
2021-01-08 PM 11:31:17 (Fri)
1/8/2021 11:31:17 PM
```

문자열 서식화 및 출력

- `Format ()` 메서드

- 문자열 보간

문자열 서식 맞추기 : 문자열 보간

- 보간: "비어 있는 부분을 채운다" 의미
 - ✓ C# 6.0 부터 지원
 - ✓ 서식항목에 첨자 대신 변수이름, 조건식 코드, 상수 등 직접 입력

문자열틀 앞에 \$ 표기 변수명 또는 반환식

\$ "텍스트{<보간식>[,길이] [: 서식]}텍스트{...}..."

서식항목에 [맞춤]

서식항목의 [서식 문자열]



문자열 보간 사용 예제 코드

```
static void Main(string[] args)
{
    Console.WriteLine("{0}, {1}", 123, "가나다");
    Console.WriteLine($"{123}, {"가나다"}"); // 정수와 문자열 입력

    Console.WriteLine("{0,-10:D5}", 123);
    Console.WriteLine($"{123,-10:D5}"); // 맞춤, 서식문자열 사용

    int n = 123;
    string s = "가나다";

    Console.WriteLine("{0}, {1}", n, s);
    Console.WriteLine($"{n}, {s}"); // 변수명 입력

    Console.WriteLine("{0}", n>100? "큼" : "작음");
    Console.WriteLine($"{(n > 100 ? "큼" : "작음")}"); // 조건연산자 사용
}
```

출력 결과:
123, 가나다
123, 가나다
00123
00123
123, 가나다
123, 가나다
큼
큼



문자열 안에서 찾기

문자열 안에서 찾기

- 문자열 안에서 지정된 문자 또는 문자열을 찾는 메소드

메소드	설명
IndexOf ()	현재 문자열 내에서 찾으려는 지정 문자 또는 문자열의 인덱스 (중복이 있을 경우 맨 앞의 것)
LastIndexOf ()	현재 문자열 내에서 찾으려는 지정 문자 또는 문자열의 인덱스 (중복이 있을 경우 맨 뒤의 것)
StartsWith ()	현재 문자열이 지정된 문자열로 시작하는지 평가 (true 또는 false)
EndsWith ()	현재 문자열이 지정된 문자열로 끝나는지 평가 (true 또는 false)
Contains ()	현재 문자열이 지정된 문자열을 포함하는지 평가
Replace ()	현재 문자열에서 지정된 문자열이 다른 지정된 문자열로 모두 바뀐 새 문자열 반환



문자열 안에서 찾기: 예제 코드

출력 결과:

```
IndexOf 'Good' : 0
IndexOf 'o' : 1
LastIndexOf 'Good' : 0
LastIndexOf 'o' : 6
StartsWith 'Good' : True
StartsWith 'G' : True
StartsWith 'Morning' : False
EndsWith 'Morning' : True
EndsWith 'M' : False
Contains 'Morning' : True
Replaced 'Morning' with 'Evening' :
Good Evening
```

```
static void Main(string[] args)
{
    string greeting = "Good Morning";

    Console.WriteLine("IndexOf 'Good' : {0}", greeting.IndexOf("Good"));
    Console.WriteLine("IndexOf 'o' : {0}", greeting.IndexOf('o'));

    Console.WriteLine("LastIndexOf 'Good' : {0}", greeting.LastIndexOf("Good"));
    Console.WriteLine("LastIndexOf 'o' : {0}", greeting.LastIndexOf('o'));

    Console.WriteLine("StartsWith 'Good' : {0}", greeting.StartsWith("Good"));
    Console.WriteLine("StartsWith 'G' : {0}", greeting.StartsWith('G'));
    Console.WriteLine("StartsWith 'Morning' : {0}", greeting.StartsWith("Morning"));

    Console.WriteLine("EndsWith 'Morning' : {0}", greeting.EndsWith("Morning"));
    Console.WriteLine("EndsWith 'M' : {0}", greeting.EndsWith('M'));

    Console.WriteLine("Contains 'Morning' : {0}", greeting.Contains("Morning"));
    Console.WriteLine("Replaced 'Morning' with 'Evening' : {0}", greeting.Replace("Morning", "Evening"));
}
```


문자열 변형하기 문자열 분할하기

문자열 변형 하기

- 문자열 변형
 - ✓ 문자열 추가 및 삭제
 - ✓ 대문자/소문자 변환
 - ✓ 문자열 앞/뒤 공백 제거

메소드	설명
ToLower ()	현재 문자열의 모든 대문자를 소문자로 바꾼 새 문자열 반환
ToUpper ()	현재 문자열의 모든 소문자를 대문자로 바꾼 새 문자열 반환
Insert ()	현재 문자열의 지정된 위치에 지정된 문자열이 삽입된 새 문자열 반환
Remove ()	현재 문자열의 지정된 위치로부터 지정된 수만큼의 문자가 삭제된 새 문자열 반환
Trim ()	현재 문자열의 앞/뒤에 있는 공백을 제거한 새 문자열 반환
TrimStart ()	현재 문자열의 앞에 있는 공백을 제거한 새 문자열 반환
TrimEnd ()	현재 문자열의 뒤에 있는 공백을 제거한 새 문자열 반환



문자열 변형 하기: 예제 코드

```
static void Main(string[] args)
```

```
{
```

```
    Console.WriteLine("ToLower() : '{0}'", "ABC".ToLower());
```

```
    Console.WriteLine("ToUpper() : '{0}'", "abc".ToUpper());
```

```
    Console.WriteLine("Insert() : '{0}'", "Happy Friday!".Insert(5, " Sunny"));
```

```
    Console.WriteLine("Remove() : '{0}'", "I Don't Love You".Remove(2, 6));
```

```
    Console.WriteLine("Trim() : '{0}'", " No Spaces ".Trim());
```

```
    Console.WriteLine("TrimStart() : '{0}'", " No Spaces ".TrimStart());
```

```
    Console.WriteLine("TrimEnd() : '{0}'", " No Spaces ".TrimEnd());
```

```
}
```

출력 결과:

ToLower() : 'abc'

ToUpper() : 'ABC'

Insert() : 'Happy Sunny Friday!'

Remove() : 'I Love You'

Trim() : 'No Spaces'

TrimStart() : 'No Spaces '

TrimEnd() : ' No Spaces'



문자열 분할 하기

메소드	설명
Split ()	지정된 문자를 기준으로 현재 문자열을 분리한 다음 분리한 문자열의 배열을 반환
SubString ()	현재 문자열의 지정된 위치로부터 지정된 수만큼의 문자로 이루어진 새 문자열을 반환

```
static void Main(string[] args)
{
    string greeting = "Good Morning";

    Console.WriteLine(greeting.Substring(0,5));
    Console.WriteLine(greeting.Substring(5));

    string[] arr = greeting.Split(" ", StringSplitOptions.None);
    Console.WriteLine("Word Count : {0}", arr.Length);

    foreach (string element in arr)
        Console.WriteLine("{0}", element);
}
```

출력 결과:

```
Good
Morning
Word Count : 2
Good
Morning
```



데이터 가공을 위한 연산자

- 산술연산자
- 증가/감소 연산자
- 문자열 결합 연산자
- 관계 연산자
- 논리 연산자
- 조건 연산자
- null 조건부 연산자
- null 병합 연산자

C#의 연산자 개요

- 각 연산자 특정 형식에 대해서만 사용 가능
 - ✓ 나눗셈 연산자 "/"는 모든 수치데이터 형식 사용 가능
 - ✓ 하지만, 나눗셈 연산자는 문자열 형식은 사용 불가

분류	연산자
산술 연산자	+, -, *, /, %
증가/감소 연산자	++, --
관계 연산자	<, >, ==, !=, <=, >=
조건 연산자	?:
null 조건부 연산자	?., ?[]
논리 연산자	&&, , !
비트 연산자	<<, >>, &, , ^, ~
할당 연산자	=, +=, -=, *=, /=, %=, &=, =, ^=, <<=, >>=
null 병합 연산자	??



산술 연산자

- 덧셈(+), 뺄셈(-), 곱셈(*), 나눗셈(/), 나눗셈의 나머지(%)
- 정수형식, 부동 소수점 형식, decimal 형식만 사용 가능

```
static void Main(string[] args)
{
    int a = 111 + 222;
    Console.WriteLine($"a : {a}");

    int b = a - 100;
    Console.WriteLine($"b : {b}");

    int c = b * 10;
    Console.WriteLine($"c : {c}");

    double d = c / 6.3;
    Console.WriteLine($"d : {d}");

    Console.WriteLine($"22 / 7 = {22 / 7}({22 % 7})");
}
```

출력 결과:

a : 333
b : 233
c : 2330
d : 369.8412698412699
22 / 7 = 3(1)

증가/감소 연산자

- 증가/감소 연산자는 변수 앞 또는 뒤에 ++/-- 추가
 - ✓ 전위 증가/감소 연산자: ++/--가 변수 앞에 위치
 - ✓ 후위 증가/감소 연산자: ++/--가 변수 뒤에 위치

연산자	이름	설명	지원 형식
++	증가 연산자	피연산자 값을 1 증가 시킴	모든 수치 데이터 형식과 열거 형식
--	감소 연산자	피연산자 값을 1 감소 시킴	모든 수치 데이터 형식과 열거 형식

```
static void Main(string[] args)
{
    int a = 10;
    Console.WriteLine(a++); // 후위 증가 연산자: 10 출력 후, a는 11로 증가
    Console.WriteLine(++a); // 전위 증가 연산자: a가 12로 증가 후, 12 출력

    Console.WriteLine(a--); // 후위 감소 연산자: 12 출력 후, a는 11로 감소
    Console.WriteLine(--a); // 전위 감소 연산자: a가 10로 감소 후, 10 출력
}
```

출력 결과:

10
12
12
10



문자열 결합 연산자

- 문자열과 문자열 사이에 "+"를 사용
 - ✓ 문자열 형식에 사용
 - ✓ 해당 두 문자열을 하나의 문자열로 연결

```
static void Main(string[] args)
{
    string result = "123" + "456";
    Console.WriteLine(result);

    result = "Hello" + " " + "World!";
    Console.WriteLine(result);
}
```

출력 결과:

123456

Hello World!



관계 연산자

- 두 피연산자 사이의 관계를 평가
 - ✓ 같은지 또는 다른지, 한쪽이 다른 한쪽보다 값이 큰지 혹은 작은지 평가
 - ✓ <, >, <=, >= 연산자 : 모든 수치 형식과 열거 형식에 사용 가능
 - ✓ ==, != 연산자 : 모든 데이터 형식에서 사용가능

```
static void Main(string[] args)
{
    Console.WriteLine($"3>4 : {3>4}");
    Console.WriteLine($"3>=4 : {3 >= 4}");
    Console.WriteLine($"3<4 : {3 < 4}");
    Console.WriteLine($"3<=4 : {3 <= 4}");
    Console.WriteLine($"3==4 : {3 == 4}");
    Console.WriteLine($"3!=4 : {3 != 4}");
}
```

출력 결과:

3>4 : False
3>=4 : False
3<4 : True
3<=4 : True
3==4 : False
3!=4 : True



논리 연산자

- 참과 거짓으로 이루어지는 진리값이 피연산자인 연산

논리곱 (&&: AND)

A	B	A && B
참	참	참
참	거짓	거짓
거짓	거짓	거짓
거짓	참	거짓

논리합 (||: OR)

A	B	A B
참	참	참
참	거짓	참
거짓	거짓	거짓
거짓	참	참

부정 (!: NOT)

A	!A
참	거짓
거짓	참

```
int a = 3;
int b = 4;
bool c = a < b && b < 5;    // c는 true
bool d = a > b && b < 5;    // d는 false
bool e = a > b || b < 5;    // e는 true
bool f = !e;                // f는 false
```



조건 연산자

- 세개의 피연산자를 사용
 - ✓ 첫 매개변수 "조건식"의 결과는 참 또는 거짓
 - ✓ 조건식이 참이면 두번째 매개변수 선택, 거짓이면 세번째 매개변수 선택
 - ✓ 두번째와 세번째 매개변수는 같은 데이터형식

사용 방법

조건식 ? 참일_때의_값 : 거짓일_때의_값

사용 예제 코드

```
int a = 30;  
string result = a == 30 ? "삼십" : "삼십아님";    // result는 "삼십"
```

조건식

참일_때의_값

거짓일_때의_값



null 조건부 연산자

- 객체의 멤버에 접근하기 전, 해당 객체가 null인지 검사
 - ✓ 객체가 null 이면, 그 결과로 null 반환
 - ✓ 객체가 null 이 아닌 경우, 뒤에 지정된 멤버를 반환
 - ✓ C# 6.0부터 지원

class Foo == 연산자를 이용한 코드

```
{  
    public int member;  
}  
  
Foo foo = null;  
  
int? bar;  
if (foo == null)  
    bar = null;  
else  
    bar = foo.member;
```

class Foo ?. 연산자를 이용한 코드

```
{  
    public int member;  
}  
  
Foo foo = null;  
  
int? bar;  
bar = foo?.member;
```



null 병합 연산자 “??”

- 두 피연산자에 대해 왼쪽 피연산자가 null인지 평가
 - ✓ 평가결과가 null 이 아니면, 그 결과로 왼쪽 피연산자 그대로 반환
 - ✓ 평가결과가 null 이면, 오른쪽 피연산자 반환

```
static void Main(string[] args)
{
    int? num = null;
    Console.WriteLine($"{num ?? 0}");

    num = 99;
    Console.WriteLine($"{num ?? 0}");

    string str = null;
    Console.WriteLine($"{str ?? "Default"}");

    str = "Specific";
    Console.WriteLine($"{str ?? "Default"}");
}
```

출력 결과:

0

99

Default

Specific



연산자의 우선순위

우선순위	종류	연산자
1	증가/감소 연산자 및 null 조건부 연산자	후위 ++/-- 연산자, ?., ?[]
2	증가/감소 연산자	전위 ++/-- 연산자
3	산술 연산자	* / %
4	산술 연산자	+ -
5	시프트 연산자	<< >>
6	관계 연산자	< > <= >=
7	관계 연산자	== !=
8	비트 논리 연산자	&
9	비트 논리 연산자	^
10	비트 논리 연산자	
11	논리 연산자	&&
12	논리 연산자	
13	null 병합 연산자	??
14	조건 연산자	?:
15	할당 연산자	= *= /= += -= <<= >>= &= ^= !=



Thank you!